A Simple Processor

with Abstract Machine Execution Exercise Solutions

1. A simple Instruction Set Architecture
2. A simple microarchitecture (implementation):
   Data Path and Control Logic

HW ISA

ex

Exercise #1:
Fill in the rest of the machine state based on this initial state

M: Data Memory

Address	Contents	
0x0 – 0x1	0x0F	0x00
0x2 – 0x3	0x04	0x01
0x4 – 0x5	0x04	0x00
0x6 – 0x7		
0x8 – 0x9		
0xA – 0xB		
0xC – 0xD		
…		

R: Register File

Reg	Contents
R0	0x0000
R1	0x0001
R2	
R3	**0x000F**
R4	**0x0104**
R5	**0x0004**
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	

PC: Program Counter

IM: Instruction Memory

Address	Contents
0x0 – 0x1	LW R3, 0(R0)
0x2 – 0x3	LW R4, 2(R0)
0x4 – 0x5	AND R3, R4, R5
0x6 – 0x7	SW R5, 4(R0)
0x8 – 0x9	HALT
…	

Processor Loop

1. *ins* ← IM[PC]
2. PC ← PC + 2
3. Do *ins*

Execution Table for *Exercise #1* (shows step-by-step execution)

Solutions

ex

PC	Instr	State Changes
0x0	LW R3 0(R0)	R[3] ← M[R[0] + 0]] = M[0] = 0x000F; PC ← PC+2 = 0+2 = 2
0x2	LW R4, 2(R0)	R[4] ← M[R[0] + 2]] = M[2] = 0x0104; PC ← PC+2 = 2+2 = 4
0x4	AND R3, R4, R5	R[5] ← R[3] & R[4] = 0x0004 ; PC ← PC+2 = 4+2 = 6
0x6	SW R5, 4(R0)	M[R[0] + 4] = M[4] ← R[5] = 0x0004; PC ← PC+2 = 6+2 = 8
0x8	HALT	*Program execution stops*

The bytes are swapped from the memory M picture on the previous page because the bytes are stored in **Little Endian** order.

E.g., for the byte pair 0x00 at address 0x0 and 0x0F at address 0x1, the byte at the lower address 0x0 is stored at the "little end" (LSB) of the 2-byte word. As we'll soon see, this is consistent with the byte ordering in the C programming language.

ex Exercise 2
Solutions

HW ISA

What is this code doing at a high level?

Multiplies the contents of R9 and R10!

PC: Program Counter

Processor Loop

1. *ins* ← IM[PC]
2. PC ← PC + 2
3. Do *ins*

M: Data Memory

Address	Contents	
0x0 – 0x1	0x0F	0x00
0x2 – 0x3	0x04	0x01
0x4 – 0x5		
0x6 – 0x7		
0x8 – 0x9		
0xA – 0xB		
0xC – 0xD		
…		

IM: Instruction Memory

Address	Contents
0x0 – 0x1	SUB R8, R8, R8
0x2 – 0x3	BEQ R9, R0, 3
0x4 – 0x5	ADD R10, R8, R8
0x6 – 0x7	SUB R9, R1, R9
0x8 – 0x9	JMP 1
0xA – 0xB	HALT
…	

R: Register File

Reg	Contents (time: →)
R0	0x0000
R1	0x0001
R2	
R3	
R4	
R5	
R6	
R7	
R8	0x???? →(1) 0x0000 →(2) 0x0003 →(4) 0x0006
R9	0x0002 →(3) 0x0001 →(5) 0x0000
R10	0x0003
R11	
R12	
R13	
R14	
R15	

Execution Table for *Exercise #2* (shows step-by-step execution)

ex

Solutions

PC	Instr	State Changes
0x0	SUB R8, R8, R8	R[8] ← R[8] – R[8] = 0; PC ← PC+2 = 0+2 = 2
0x2	BEQ R9, R0, 3	PC ← PC+2 = 2+2 = 4 (because 2 = R[9] ≠ R[0] = 0)
0x4	ADD R10, R8, R8	R[8] ← R[10] + R[8] = 3 + 0 = 3; PC ← PC+2 = 4+2 = 6
0x6	SUB R9, R1, R9	R[9] ← R[9] - R[1] = 2 – 1 = 1; PC ← PC+2 = 6+2 = 8
0x8	JMP 1	PC ← 2*1 = 2
0x2	BEQ R9, R0, 3	PC ← PC+2 = 2+2 = 4 (because 1 = R[9] ≠ R[0] = 0)
0x4	ADD R10, R8, R8	R[8] ← R[10] + R[8] = 3 + 3 = 6; PC ← PC+2 = 4+2 = 6
0x6	SUB R9, R1, R9	R[9] ← R[9] - R[1] = 1 – 1 = 0; PC ← PC+2 = 6+2 = 8
0x8	JMP 1	PC ← 2*1 = 2
0x2	BEQ R9, R0, 3	PC ← PC+2+(2*3) = 4+6 =10 (because 0 = R[9] = R[0] = 0)
0xA	HALT	*Program execution stops*